Evaluating the impact of caching on the energy consumption and performance of progressive web apps

Malavolta, Ivano; Chinnappan, Katerina; Jasmontas, Lukas; Gupta, Sarthak; Soltany, Kaveh Ali Karam

published in
MOBILESoft '20
2020

DOI (link to publisher)
10.1145/3387905.3388593

document version
Publisher's PDF, also known as Version of record

Evaluating the Impact of Caching on the Energy Consumption and Performance of Progressive Web Apps

Ivano Malavolta, Katerina Chinnappan, Lukas Jasmontas, Sarthak Gupta, Kaveh Ali Karam Soltany
Vrije Universiteit Amsterdam, The Netherlands
i.malavolta@vu.nl, {k.p.chinnappan | l.jasmontas | s3.gupta | k.alikaramsoltany}@student.vu.nl

ABSTRACT

Context. Since today mobile devices have limited battery life, the energy consumption of the software running on them can play a strong role with respect to the success of mobile-based businesses. Progressive Web Applications (PWAs) are built using common web technologies like HTML, CSS, and JavaScript and are commonly used for providing a better user experience to mobile users. Caching is the main technique used by PWA developers for optimizing network usage and for providing a meaningful experience even when the user's device is offline.

Goal. This paper aims at assessing the impact of caching on both the energy consumption and performance of PWAs.

Method. We conducted an empirical experiment targeting 9 real PWAs developed by third-party developers. The experiment is designed as a 1 factor - 2 treatments study, with the usage of caching as the single factor and the status of the cache as treatments (empty vs populated cache). The response variables of the experiment are (i) the energy consumption of the mobile device and (ii) the page load time of the PWAs. The experiment is executed on a real Android device running the Mozilla Firefox browser.

Results. Our results show that PWAs do not consume significantly different amounts of energy when loaded either with an empty or populated cache. However, the page load time of PWAs is significantly lower when the cache is already populated, with a medium effect size.

Conclusions. This study confirms that PWAs are promising in terms of energy consumption and provides evidence that caching can be safely exploited by PWA developers concerned with energy consumption. The study provides also empirical evidence that caching is an effective technique for improving the user experience in terms of page loading time of PWAs.

CCS CONCEPTS

• **Software and its engineering → Software performance**; • **Information systems → Web applications**.

ACM Reference Format:
Ivano Malavolta, Katerina Chinnappan, Lukas Jasmontas, Sarthak Gupta, Kaveh Ali Karam Soltany. 2020. Evaluating the Impact of Caching on the Energy Consumption and Performance of Progressive Web Apps. In *IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems (MOBILESoft '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3387905.3388593

MOBILESoft '20, October 5–6, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7959-5/20/05…$15.00
https://doi.org/10.1145/3387905.3388593

1 INTRODUCTION

Introduced for the first time in 2015, Progressive web apps (PWAs) are special kinds of mobile web apps supporting, among others, progressive enhancement, low or no network connectivity, background processing capabilities, and push notifications [19]. The essence of a PWA is that it behaves like a typical mobile application without the need of uploading it onto Google Play or Apple App Store. PWAs do not require stable network conditions and they tend to work faster than standard mobile-optimized websites [2]. At the core of PWAs lie service workers[1] , i.e., a set of JavaScript APIs for allowing developers to programmatically cache and preload assets and data, perform background operations, receive push notifications, etc. One of the main features of PWAs and one of the main reasons why they have been gaining popularity, is that they allow developers to provide the so-called *offline-first experience* [26]. Specifically, when a standard web app is launched on a mobile device, the user must be on-line and wait for the web page to load, including the dynamic contents of the page, images, style-sheets, scripts, etc. Differently, PWAs can store resources and JavaScript modules in a dedicated cache in the browser once they are first accessed, allowing the PWA to work also when the user is offline.

If on one side caching improves user experience, reliability, and performance [1], on the other side service workers require that their JavaScript source code is parsed, downloaded, and executed, implying the usage of additional computational resources, which in turn may impact the energy consumption and performance of the PWA as a whole.

The **goal** of this study is to empirically assess the impact of *caching* on both the *energy consumption* and *performance* of PWAs. We achieve this goal by conducting an empirical experiment targeting 9 real PWAs developed by third-party developers. The experiment is designed as a 1 factor - 2 treatments study, with the usage of caching as the single factor and the status of the cache as treatments (empty vs populated cache). The response variables of the experiment are (i) the energy consumption of the mobile device and (ii) the page load time of the PWAs. The experiment is executed on a real Android device running the Mozilla Firefox browser.

The main **contributions** of this paper are:

- the results of an experiment on the impact of caching on the energy efficiency and performance of 9 real progressive web apps;
- a discussion of the obtained results and their implications for developers;

[1] http://www.w3.org/TR/service-workers/

- the replication package of our experiment containing its subjects, execution scripts, analysis scripts, and raw data.

The results of this study benefit both (i) mobile web developers in better understanding to which extent caching plays a role in terms of energy consumption and performance of their PWAs, (ii) browser vendors (e.g., Mozilla) with objective evidence about caching in PWAs, which may be used for taking better informed decisions when designing/improving the caching engines of their browsers.

The remainder of this paper is organized as follows. Section 2 provides background information about caching in PWAs. Sections 3 and 4 present the experiment design and its execution infrastructure, respectively. Section 5 describes the results of the experiment, followed by a discussion in Section 6. Threats to validity and related work are presented in Sections 7 and 8, respectively. Section 9 closes the paper.

2 BACKGROUND

Caching in mobile networks is important to compete with the unprecedented growth of network traffic [21]. Network caching is defined as the technique to store information which can be reached recurrently from the origin of the request. It helps in decreasing web traffic and the website load time for the users [3]. In addition, web caching is the short-term storage mechanism employed in network caching with the main function of delivering the webpage content to end-users as fast as possible. Distributed networks, with multiple servers, save copies of web content which are cached and retrieved from the closest server based on user requests [23]. In a recent study conducted by Qian et al. [23] on web caching in smartphones, it was concluded that web downstream traffic generated by mobile browsers and apps exceeds all other types of network traffic [11]. There exists a major gap between specification and implementation of the protocol for web caching in mobile devices today. Reviewing and fixing this could in turn drastically reduce the network traffic caused by mobile devices, along with the resources and energy consumed. Optimization of the caching parameter configurations, cache validation, delta encoding and utilization of the offline application cache are some of the most prominent ways that could help in reducing mobile network overheads and the delays or wait time of the end users [23].

In PWAs, caching is done with the help of *service workers*. A service worker is a JavaScript script that is part of the PWA, with the exception that it is executed in its own thread (instead of running on the main thread of the web app) and the browser runs it in background [13]. There are four main caching strategies commonly applied in PWAs today [10]. The first one, *Stale-While Revalidate* strategy, is used mostly for assets with a certain tolerance. It would return a cached version of the resource immediately, while it checks for an updated version on the network. If the updated version is found, it is saved to the cache for subsequent requests. The second strategy used is *Cache-Only* which, as the name suggests, returns the cached version of the resource without checking for an updated version on the network. This works well on pre-cached static resources since it only updates the contents when the application version is updated itself. *Network-First* is the third strategy which falls back on a cached version if it fails to fetch the latest contents from the network server and in turn, helps to keep the dynamic resources updated. The fourth strategy is a *Cache-First* strategy which checks the network for the latest version to update the resource in cache, but only if the first fetch from the cache fails. This helps in optimizing repetitive asset requests, as it only hits the network server for updated resources.

```
1 // required files
2 workbox.precaching.precacheAndRoute([
3     '/css/app.css',
4     '/js/offline_page.js',
5     'img/logo.png',
6     '/img/default_thumb.png'
7     ]);
8 var networkFirstHandler = workbox.strategies.
      networkFirst({
9   cacheName: "default",
10  plugins: [
11    new workbox.expiration.Plugin({
12      maxEntries: 10
13    }),
14    new workbox.cacheableResponse.Plugin({
15      statuses: [200]
16    }),
17  ],
18 });
19 // Handle all navigation requests - can
20 // switch to offline mode
21 const navMatcher = ({ event }) => event.request.
      mode === "navigate";
22 const navHandler = args =>
23   networkFirstHandler
24     .handle(args)
25     .then(response => (!response ? caches.match("/
           offline.html") : response));
26 workbox.routing.registerRoute(navMatcher,
      navHandler);
```

Listing 1: Example of service worker from a real PWA

As a concrete example, Listing 1 illustrates a fragment of the service worker containing the caching mechanism of one of the PWAs used in this study (`slate.com`). The listing uses *Workbox*, a collection of JavaScript libraries for PWAs provided by Google[2]. Firstly, all necessary static resources to be cached are defined in an array and stored in a precaching data structure (lines 2-7). Precaching consists in the process of adding files to the cache of the browser when the service worker loads the first time. It is important to note that cacheable resources can be of any type, including for example CSS stylesheets, JavaScript files, images. Then, the "network first" strategy is initialized in order to allow cacheable resources to come from the network whenever possible, but also to fallback to their cached version if the network fails (lines 8-18); here the developers specified also that the maximum number of entries in the cache is 10 (line 12) and that every server response with status code `200` is cacheable. Finally, the service worker defines and registers a handler for navigation events in the PWAs such that a static `offline.html` page is shown to the user in case their mobile device is currently offline (lines 21-26).

3 EXPERIMENT DESIGN

3.1 Goal and Research Questions

Intuitively, the goal of the experiment is to evaluate the impact caching has on both the energy consumption and the performance

[2]https://developers.google.com/web/tools/workbox

of PWAs. More specifically, we will investigate how having an empty or already-populated cache will impact the energy consumption and performance of a PWA. Table 1 shows a more formal definition of the goal using the Goal-Question-Metric technique [4, 6].

Table 1: Goal definition

Analyze	Cache status
For the purpose of	Evaluating its *impact* on
With respect to	Energy consumption and performance
From the point of view of	Developers and researchers
In the context of	Real-world PWAs
Combination	
Analyze the cache status for the purpose of evaluating its impact on the energy efficiency and performance from the point of view of developers and researchers in the context of real-world PWAs	

The goal described above is refined into the following two research questions.

[RQ1]: *How does the cache status impact the energy consumption of progressive web apps?* This research question is asked to assess and evaluate the impact of the status of the browser's cache on the *energy consumption* of PWAs.

[RQ2]: *How does the cache status impact the performance of progressive web apps?* The purpose of this research question is to assess and evaluate the impact of empty and populated cache on the *performance* of progressive web applications.

Answering these research questions will help developers and researchers to better understand caching and its impact in PWAs, and utilize that knowledge to identify and repair energy hotspots in their apps more effectively in the future.

3.2 Subjects Selection

In order to make our experiment representative of real development practices, we decided to use *real-world* PWAs developed by third-party developers as subjects. To this aim, we consider three different collections of third-party PWAs as starting point of our subjects selection phase, namely: PWA Rocks[3], Awesome PWA[4], and a recently published on-line article on a technical blog[5]. The considered collections are well-known in the web community, actively maintained, and several independent members of the community are contributing to them. We mined all PWAs mentioned in the three collections and combined them into a single source of data (while removing duplicates). This resulted in a collection of 100 unique PWAs, which could potentially be used as subjects for our experiment. Then, we locally downloaded all the frontend-related software artifacts of each PWA, i.e., HTML, CSS, and JS code, but also images and other static resources. This step has been done for all 100 PWAs and it has been performed via a third-party Google Chrome extension[6].

[3]https://pwa.rocks/
[4]https://github.com/hemanth/awesome-pwa
[5]https://www.tigren.com/examples-progressive-web-apps-pwa
[6]https://github.com/up209d/ResourcesSaverExt

We perform a further selection process with the aim of identifying a set of PWAs which are representative of the population of *real-world* PWAs (e.g., discarding toy examples, PWAs with only a login screen, videogames). Inspired by the systematic literature review methodology [28], we first defined a priori a set of inclusion and exclusion criteria, then we *manually analyzed* each potentially relevant PWA and selected it according to the selection criteria. A PWA was selected if it satisfied *all* inclusion criteria and *none* of the exclusion criteria. Table 2 shows the selection criteria used in this study.

Table 2: Subjects inclusion and exclusion criteria

Inclusion criteria	
I1	The PWA is *data-driven* (the most recurrent application scenario for PWAs [20]), i.e., it shows data depending on the interaction events of the end user, where data is typically provided by back-end services [12]
I2	The PWA shows dynamic data at load time (e.g., no landing pages)
I3	The locally downloaded frontend of the PWA loads properly on a mobile device (i.e., no missing images, no broken links, etc.)
Exclusion criteria	
E1	The PWA is admittedly experimental or in an alpha/beta stage
E2	The PWA is a videogame (videogames have a totally different development model of standard data-driven PWAs)
E3	The PWA shows only a login screen at startup time

Table 3 reports the selected PWAs, together with their URL and category. The subjects of this study are heterogeneous in terms of size and category, making us reasonably confident about their representativeness of the targeted population.

Table 3: Subjects of this study

Progressive Web App	Category
petlove.com.br	Shopping
m.alibaba.com	Shopping
nylon.com	Magazine
zumata.com	Business
slate.com	News
smashingmagazine.com	Magazine
edgy.app	News
soundslice.com	Entertainment
nrw-tourismus.de	Entertainment

3.3 Experimental Variables

For answering the defined research questions in Section 3.1, we consider the **cache status** as our independent variable, with two treatments: *empty cache* and *populated cache*. At every experiment run, the *empty cache* treatment is enforced by completely cleaning the Firefox mobile app running on the mobile device and then loading the PWA, whereas the *populated cache* treatment is enforced

by launching the PWA twice within the same run while measuring the dependent variables only during the second launch.

The dependent variables of the experiment are two (one for each research question):

- **Energy consumption**: measured in Joules (J), it represents the amount of energy consumed by the Mozilla Firefox browser running on the mobile device for fully loading the PWA (see below);
- **Page load time**: measured in milliseconds (ms), it represent the amount of time between the initial HTTP `GET` request issued from the Mozilla Firefox browser running on the mobile device and the moment in which the browser raises the *load*[7] event.

3.4 Experimental Hypotheses and Design

We have formulated the following hypotheses in order to answer the research questions of our study.

Given that μ_{empty} is the average energy consumption of PWAs launched with an empty cache and $\mu_{populated}$ is the average energy consumption of PWAs launched with a populated cache, then the null and alternate hypotheses for RQ1 are defined as:

$$H_0^e: \mu_{empty} = \mu_{populated}$$
$$H_1^e: \mu_{empty} \neq \mu_{populated}$$

Similarly, given that ρ_{empty} is the average page load time of PWAs launched with an empty cache and $\rho_{populated}$ is the average page load time of PWAs launched with a populated cache, then the null and alternate hypotheses for RQ2 are defined as:

$$H_0^p: \rho_{empty} = \rho_{populated}$$
$$H_1^p: \rho_{empty} \neq \rho_{populated}$$

Based on the subjects, variables, and hypothesis of our experiment, we designed our experiment as a balanced 1 factor - 2 treatments experiment with a *crossover design* [28], i.e., each PWA receives both the *empty cache* and the *populated cache* treatments across experimental runs. In order to take into account the intrinsic variability of energy measurement in mobile devices [15], each trial of our experiment has a *repeated measures design*, where the measures of both energy consumption and page load time are collected 20 times for each subject-treatment combination. Finally, the measurements executions are randomized in order to take into account possible contextual variations within the measurement infrastructure (see Section 4).

3.5 Data Analysis

The data obtained from the experiment runs is analyzed quantitatively. Firstly, for each dependent variable we will investigate on the *normality* of its distribution by (i) visually inspecting the distribution of the obtained measurements, (ii) building a Q-Q plot against the theoretical normal distribution and visually inspecting it, and (iii) applying the *Shapiro Wilk* test, where the null hypothesis is that the data comes from a normal distribution.

Then, if the data will be normally distributed we will apply the *paired t-test* statistical test to test the statistical hypothesis related to each research question. Moreover, in case the obtained data is not normally distributed, as suggested in [27], we will *transform* measurement data in order to explore the possibility of having a normal distribution, which can potentially lead to higher statistical power. If also the transformed data will not follow a normal distribution, then the Wilcoxon signed-rank non-parametric test will be used since it does not make any assumptions about the data being analyzed. All statistical tests will be performed with $\alpha = 0.05$ as significance level.

Finally, in order to statistically assess the magnitude of the impact of caching, we apply the Cliff's Delta statistical test to both the energy consumption and the page load time [8]. The Cliff Delta is a non-parametric effect size for ordinal variables and it does make any assumptions about the distributions being compared. The values of the obtained Cliff Delta measures are interpreted according to the guidelines proposed by [14].

3.6 Study Replicability

In order to foster independent verification and replication of this experiment, a full replication package is publicly available on GitHub[8]. The replication package contains (i) the full list of potentially usable subjects (before our application of inclusion/exclusion criteria), (ii) the Python scripts for locally downloading the frontends of the potential subjects, (ii) the raw data containing all the measures collected during the execution of the experiment, (iii) the R scripts for analysing the obtained data, and (iv) a guide containing the steps for replicating the experiment.

4 EXPERIMENT EXECUTION

We developed a dedicated tool chain for automating several steps of the execution of the experiment. In the following we describe our tool chain, together with third-party libraries and tools we used to achieve the goal of reliably measuring the energy consumption and page load time of the 9 PWAs described in Section 3.2. Specifically, for orchestrating the execution of all the runs of the experiment we make use of Android Runner [24]. Android Runner is a Python framework for automatically executing experiments involving both native and web application running on Android-based devices. In Android Runner experiments are defined in a descriptive manner as a JSON file, and then the full execution of the experiment is managed by the tool via a combination of Python scripts and Android Debug Bridge (ADB[9]) commands.

Table 4: Technical characteristics of the mobile device

Feature	Specifications
OS	Android 9.0 (Pie)
Chipset	Qualcomm SDM845 Snapdragon 845 (10 nm)
CPU	Octa-core (4x2.5 GHz & 4x1.6 GHz Kryo 385)
GPU	Adreno 630
Memory	4GB
Display	1080x2160 pixels, 5.5" POLED capacitive touchscreen

[7] https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event

[8] https://github.com/S2-group/mobilesoft-2020-caching-pwa-replication-package

[9] https://developer.android.com/studio/command-line/adb

In our experiment, Android Runner is executed on a laptop with Ubuntu 18.04.1 LTS and 4GB of memory. For this experiment we use a real mobile device with the technical characteristics listed in Table 4. PWAs are hosted on the laptop, served over a Wifi network, and executed within an instance of the Mozilla Firefox browser (version 68.1.1). For this experiment we use a real mobile device with the technical characteristics listed in Table 4.

Both the laptop and the mobile device run under the same WiFi network with a speed of 100 Mbps. To ensure that the WiFi conditions do not alter the results of the experiment, the mobile device and the laptop are the only devices connected to the WiFi network. Further, we take special care in keeping the execution environment as clean as possible, specifically: the mobile device is loaded with a clean installation of the Android OS, it has been configured so to do not perform any OS updates, Google services have been disabled, all third-party apps have been uninstalled, and push notifications have been disabled.

With the laptop connected to mobile device, each run of the experiment is executed by automatically launching each PWA in the Firefox app running on the device, while its energy consumption and page load time are measured. To measure the *energy consumption* of the PWAs we use the `dumpsys`[10] tool; it is maintained by Google and it is generally used for programmatically providing information about the system services running on an Android device. When running `dumpsys` with the `batterystats` option, it generates data about battery usage on a device, such as the power use per app and system component, the history of battery-related events, etc. The *page load time* metric is obtained by collecting the timestamp of the beginning of the launch of each PWA and the timestamp of the full load event of the PWA. The launch timestamp is collected by the orchestration tool at the beginning of every run of the experiment, whereas the page load timestamp is obtained by injecting a snippet of JavaScript code into the `index.html` file of each PWA which listens to the `load` event triggered by the PWA and sends it back to the orchestration tool via ADB.

In order to take into account the intrinsic variability of energy and page load time measurements, we take the following precautions: (i) the order of execution of the experiment runs is randomized, (ii) the measurement of each PWA is repeated 20 times, (iii) between each run the mobile device remains idle for 60 seconds so to take into account tail energy usage, *i.e.,* the phenomenon where certain hardware components of mobile devices are optimistically kept active by the OS to avoid startup energy costs [17], and (iv) depending on the specific treatment to be considered, the Firefox app is cleared before each run so to reset its cache and persisted data.

5 RESULTS

This section presents the results of our empirical experiment for each research question.

5.1 Impact on Energy Consumption (RQ1)

The descriptive statistics of the collected energy measures across all subjects and treatments are reported in Table 5. The total energy consumed to load a PWA is between 1.719 joules and 26.613 joules. A standard deviation of 1.761 shows that the collected energy measures are quite concentrated around the mean, which has a value of 5.124 for our dataset. Also, the low standard deviation is an indication of the reliability of our measurement infrastructure which indeed presents very few outliers.

Table 5: Descriptive statistics of the collected energy measures

	Energy (J)
Minimum	1.719
1st Quantile	4.094
Median	5.094
Mean	5.124
3rd Quantile	6.154
Maximum	26.613
Standard deviation	1.761

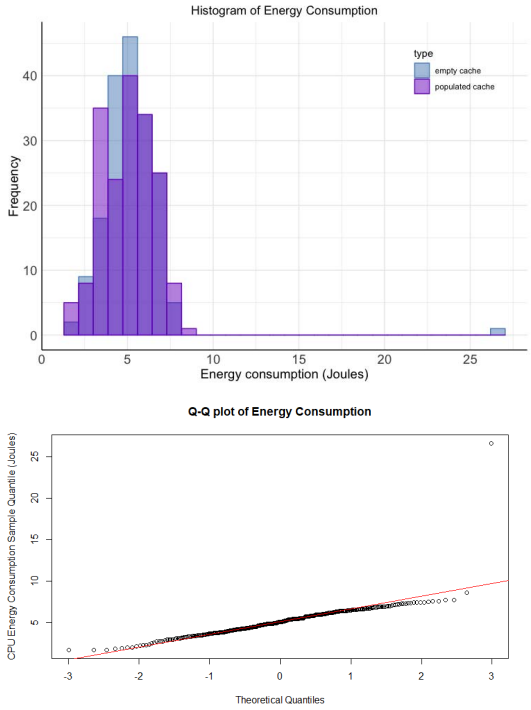

Figure 1: Histogram and Q-Q-plot of energy consumption measures

Figure 1 illustrates a Q-Q-plot and histogram depicting the energy consumption of PWAs with either an empty or populated cache. The histogram in Figure 1 depicts a bell-curve hinting on normal distribution of the data. However, an outlier with energy consumption with the value of 26.6 is present. The outlier value of 26.6 comes from the *petlove.com.br* subject. The skewness reports a positive value of 4.908, thereby implying that the data distribution is skewed to the right. The Q-Q-plot in the same figure for energy consumption depicts the extreme outlier as well. In order to avoid

[10] https://developer.android.com/studio/command-line/dumpsys

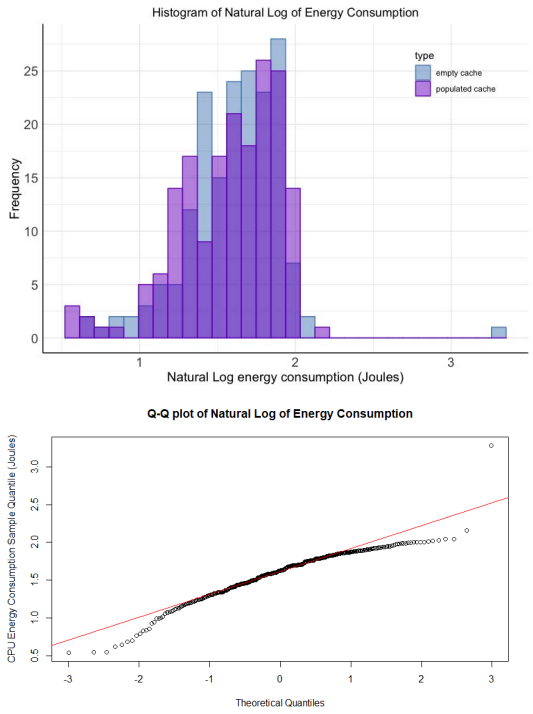

Figure 2: Log-transformed energy consumption measures

bias, we decided to do not remove the mentioned outlier. Moreover, we performed the Shapiro-Wilk test for normality and the outcome was a p-value of $3.212e-14 < \alpha$, meaning that the null hypothesis H_0 that the data comes from a normal distribution was rejected.

We performed a number of transformations to the obtained data in order to check if we could obtain a normally distributed dataset, and thus being able to apply more powerful statistical tests. In Figure 2 we show the results of a log transformation of the energy consumption measures. However, also the log-transformed data does not follow a normal distribution (the details about the performed tests are available in the replication package).

Since the energy consumption measures are not following a normal distribution, we use the Wilcoxon signed-rank non-parametric test to determine if there is a statistically significant difference between the energy consumption of the PWAs across the two treatments of the *cache status* factor (i.e., empty and populated cache). The test yields a p-value of 0.489, thereby, we are unable to reject the null H_0^e hypothesis. This means that we are not able to claim that PWAs with either an empty or populated cache consume different amounts of energy on our mobile device.

5.2 Impact on Performance (RQ2)

The descriptive statistics in Table 6 are about the page load time of all the considered PWAs, either with empty or populated cache. The page load time across all PWAs varies from 0.089 seconds to a maximum of 5.241 seconds. The highest value is 5.241 and it comes from the *nrw-tourismus.de* subject. It is interesting to note that the energy consumption outlier (26.6) belongs to a different PWA (*petlove.com.br*). In Table 6 we can also note a low standard deviation (0.610), again indicating a high tendency of the collected page load times towards their mean value (0.6758).

Table 6: Descriptive statistics of the collected page load times

	Page load time (s)
Minimum	0.089
1st Quantile	0.351
Median	0.547
Mean	0.676
3rd Quantile	0.726
Maximum	5.241
Standard deviation	0.610

Figure 3 illustrates a Q-Q-plot and histogram depicting the page load time of all PWAs for both an empty and populated cache. The skewness of the page load times reports a positive value of 2.810475, which can be seen also in the histogram in Figure 3, with the data distribution skewed to the right. In the same figure, the Q-Q-plot for the page load times also shows that the data is skewed to the right, implying that the data may not be normally distributed. Similarly to what we did for energy consumption measures, we apply the Shapiro-Wilk test for normality to the collected page load times. The obtained p-value is $1.404e-07 < \alpha$, allowing us to reject the null hypothesis that the data follows a normal distribution.

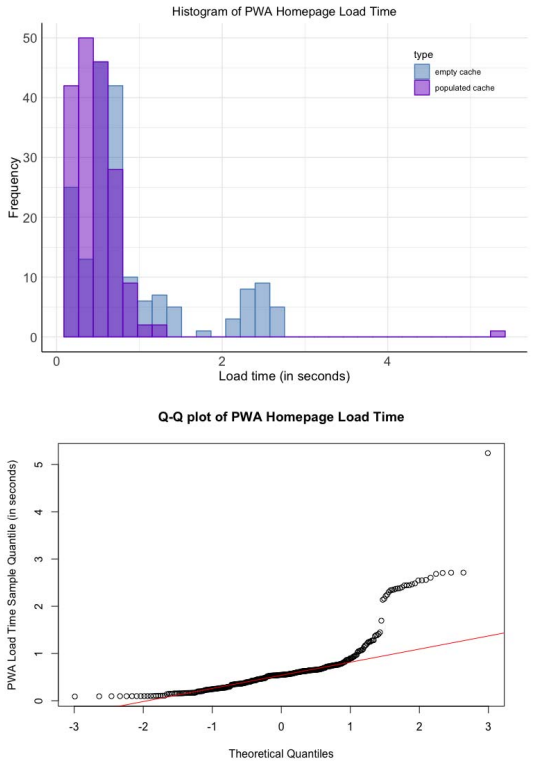

Figure 3: Histogram and Q-Q-plot of PWA page load times

Figure 4 illustrates the log transformation of the PWA page load times. It can be observed that the skewness is much lower in the histogram of the log-transformed data. However, the Q-Q plot still tends to show not normally distributed data and after applying again the Shapiro-Wilk test, we obtain a p-value $1.404e-07 < \alpha$, making us reasonably confident that the data collected for the page load times of our PWAs does not follow a normal distribution.

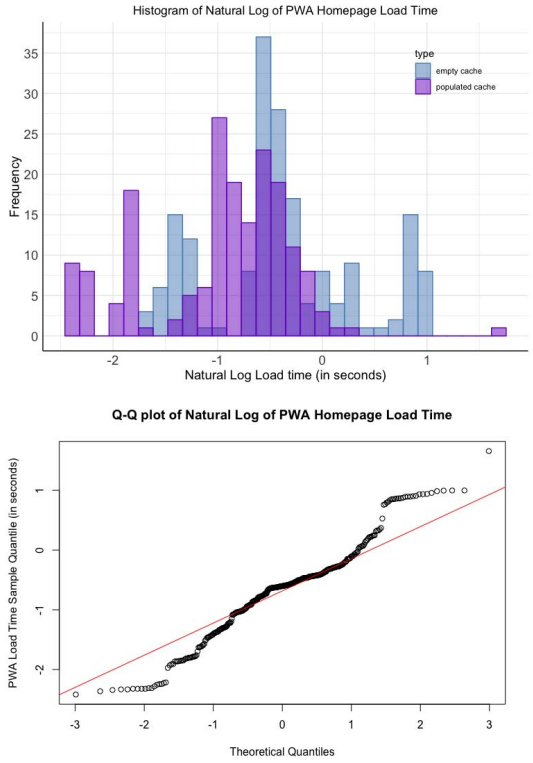

Figure 4: Log-transformed page load times

Since also the distribution of the page load times is not normal, we again use the Wilcoxon signed-rank non-parametric test to determine if there is a statistically significant difference between the page load times of PWAs with either an empty or populated cache. The p-value reported by the test is $2.2e-16 < \alpha$. Thus, we are able to reject the H_0^p null hypothesis, and claim that the status of the browser cache has an impact on the performance of PWAs. So far, we are able to prove that caching plays a role on the performance of a PWA in terms of page load time; however, in order to find out *how much* is such an impact, now we compute the effect size of the phenomenon we just discovered. To do so, we apply the Cliff's delta effect size measure to the data related to the page load times of the PWAs. The obtained Cliff's delta effect size measure is -0.419, thus revealing a *medium* effect size ($0.33 \leq |d| \leq 0.47$) in favor of the *populated cache* treatment. The obtained result provide objective evidence about the fact that loading a PWA with a populated cache in the browser leads to faster load times, with a medium effect on them.

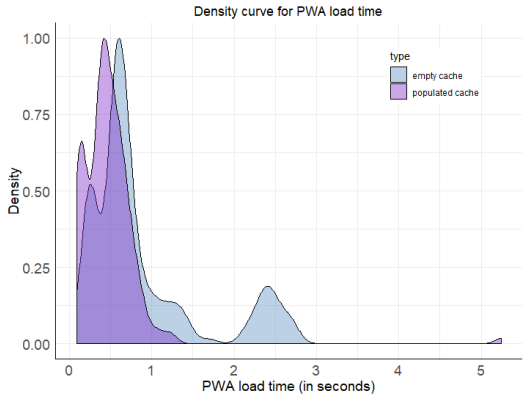

Figure 5: Density of page load times with empty and populated cache

To further support and look deeper into the results obtained with the Cliff's delta measure, in Figure 5 we show the density plot of the page load times with empty and populated cache. We can visually confirm that the difference between the page load times under the two different treatments (empty cache and populated cache) is existing and that PWAs loaded with an empty cache tend to load slower than PWAs loaded with an already populated cache.

6 DISCUSSION

The results obtained from our experiment are very different if we consider energy consumption and page load times separately. Specifically, energy consumption does not significantly differ when loading a PWA with either an empty or a populated browser cache, with an average energy consumption of 5.124 Joules (see Section 5.1). The insight that we can get from this result is that the energy consumed for the additional network requests required by a PWA with an empty cache is compensated by the (mostly I/O) operations performed by the browser and the PWA for checking cache hits and retrieving the previously cached elements. Another possible reason may be that the involved physical components may not consume energy proportional to time. This finding is even more surprising (and should raise a warning to both browser vendors and developers) if we consider that page load time is statistically lower when loading PWAs with an already-populated browser cache, with a medium effect with respect to an empty browser cache (see Section 5.2). Indeed, if we recall that $energy = power \times time$, in principles and with comparable conditions, having longer execution times would lead to higher energy consumption; this result stresses the fact that the operations performed by the browser and the PWAs for managing their cache are highly demanding in terms of *power*. Browser vendors like Mozilla can minimize the energy consumption of their products by further investigating on and optimizing the amount of power needed for managing the caching mechanisms operated by their implementation of the service worker W3C standard.

In order to better understand and provide additional insights about the previously discussed results, we analysed the energy consumption and page load time of each single PWA across both

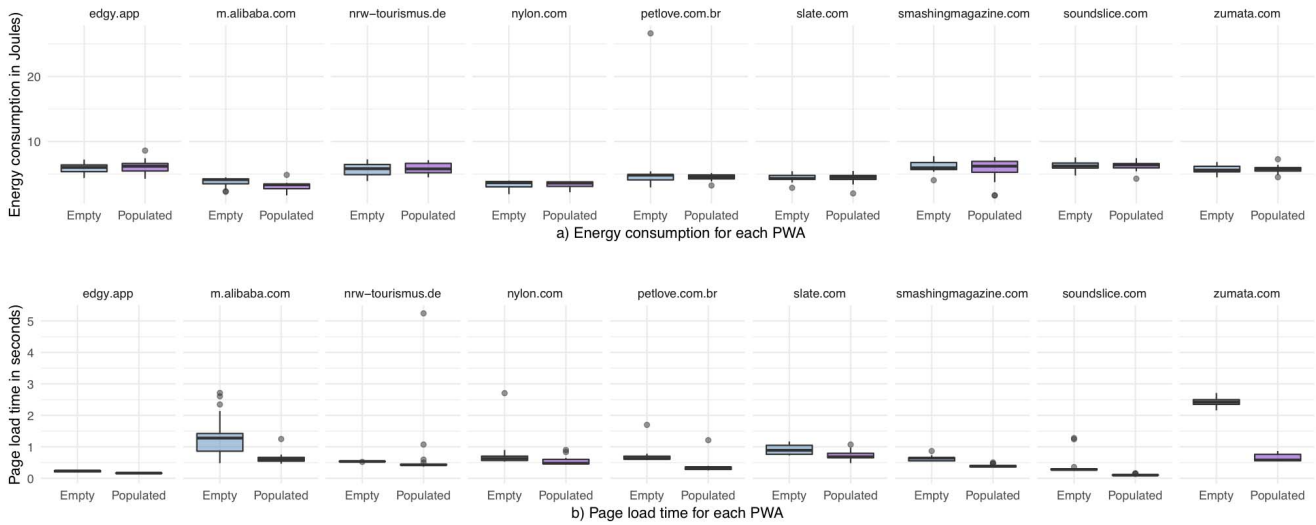

Figure 6: Energy consumption and page load time for each PWA

the *empty cache* and *populated cache* treatments. Figures 6(a) and 6(b) show the obtained results. Despite some sporadic outlier, both figures show that the results of each PWA are predominantly in line with the global results obtained by looking at all PWAs altogether. Two notable exceptions are about the page load time of the *m.alibaba.com* and *zumata.com* subjects, whose page load times are *considerably* longer when they are launched with an *empty* cache. As a first indication, we checked the overall size of each of the 9 subjects, but both *m.alibaba.com* (5.2Mb) and *zumata.com* (2.2Mb) are in line with the size of all the other subjects (average=4.14Mb). We speculate that the exceptionality of *m.alibaba.com* and *zumata.com* is related to how cacheable resources are managed in their first load, but further detailed analysis should be dedicated to get solid evidence about how and why those two cases have larger differences. For example, the service worker of *m.alibaba.com* has 8410 lines of JavaScript code (after beautification); this is exceptional even for service workers used in production [20] and it may indicate that the caching mechanism of *m.alibaba.com* is particularly advanced, potentially resulting in a more effective caching strategy.

7 THREATS TO VALIDITY

We have analyzed 4 different kinds of threats to validity [29] for the purpose of evaluating the soundness of our results.

7.1 Internal Validity

Threats to internal validity are strongly related to the experiment design and execution.

7.1.1 ***History.*** We considered two options on how to execute the experiment and collect the results. One of the options was to collect the results in an incremental manner. A total of 20 repetitions were needed to be performed, with each repetition pertaining to one of the 18 trials per subject (2 runs per PWA). We considered to perform a subset of the runs of the experiment per day, with the total time span for experiment execution of several days. The second option was to execute all 20 repetitions for all trials and collect the results in one shot. We have opted for the second option, by ensuring that all results were produced under the same setting and conditions. We have written a script to automate the repetitions 20 times for each trial, so that no manual work needed to be done. Before the actual execution of all experiments, we produced trial runs to ensure that the testing environment was reliable and no interruptions occurred during the execution phase. The motivation behind going for the second option was to assure that a reliable set of results was produced for each experiment under the same conditions and thereby, mitigate this threat.

7.1.2 ***Maturation.*** In order to mitigate this threat, we have set the time interval in Android runner in between the runs to 60 seconds. Two runs were performed for each PWA - first run with an empty cache, and second run with a populated cache. We have cleared the cache before each subsequent PWA, to ensure that the cache was populated only with the contents of the current PWA being launched.

7.1.3 ***Selection.*** Two iterations were performed in selecting the subjects before finalizing the chosen subjects for the experiment. The first iteration was performed in a random order; after collecting 100 PWAs, 10 PWAs were selected from the list in a random manner. However, it was evident that some of the PWAs did not load properly on the mobile device via the localhost server. The second iteration was performed manually, in order to ensure that all selected PWAs loaded properly on the mobile device and there were no complications. The final set of subjects consisted of 9 chosen PWAs. Selection might play a role in this experiment since the selection process of the subjects was done manually and the selected group could possibly not represent the population; however, in order to mitigate this threat, we have performed a rigorous selection procedure during the PWAs selection process (see Section 3.2),

therefore discarding non-data-driven PWAs, toy examples, PWAs behind login walls, etc.

7.1.4 ***Reliability of measures.*** Factors such as screen brightness, network connectivity, and other kinds of interruptions can affect the reliability of the measures. To mitigate this threat, we have set the brightness of the screen of the mobile device to a minimum value, performed all experiment runs in one round, and placed the mobile device near the WiFi router so to ensure that the network connectivity was constant for each run of the experiment.

7.2 External Validity

Threats to external validity are conditions that limit the ability to generalize results.

7.2.1 ***Interaction of setting and treatment.*** This threat corresponds to using unrealistic environment settings and resources. To mitigate this threat, we used a relatively modern mobile device and connected it via WiFi - a usual setting when users browse PWAs. We used the Mozilla Firefox browser, as it was not possible to install a different version of Google Chrome on our mobile device since it was branded by Google and the installed version of Chrome on the device was not suitable for the experiment.

7.2.2 ***Interaction of history and treatment.*** This threat deals with the fact that the experiment was executed on a special day at a special time, which could affect the results. To mitigate this threat, the experiment was executed on a weekday during working hours, to simulate a more realistic testing environment.

7.3 Construct Validity

A threat to the construct validity concerns the relationship between the theory and observations, and generalizing the observations.

7.3.1 ***Inadequate preoperational explication of constructs.*** We defined our constructs *a priori*, prior to the experiment execution, thereby, mitigating this threat. We used the GQM approach to define our goal, which then guided the definition of the research questions of this study. The hypotheses, dependent and independent variables, and treatments were all defined during the planning phase of the experiment.

7.3.2 ***Mono-operation bias.*** Even though we had one factor - cache, which was the independent variable, to mitigate this threat and to provide a full picture of the theory, we performed a total of 20 repetitions per trial, resulting in a total of 360 independent runs within the whole execution of the experiment.

7.3.3 ***Measurement of page load only.*** In this study we investigated only the *loading* of PWAs, i.e., we collected measures within the time span ranging from the first HTTP `GET` request to the targeted subject and its full load (proxied by the triggering of the `load` event). The rationale for this decision is to have a fully controllable environment with objectively-measurable metrics. However, if a subject is using a network-first caching strategy, it is possible that the impact of caching could reveal itself only when the user is accessing some specific functionalities, potentially leading to completely different results. As a future work, we are planning to expand this study in order to execute realistic usage scenarios for each subject in each run of the experiment, instead of focussing on page load only.

7.4 Conclusion Validity

Threats to conclusion validity deal with issues concerning the ability to determine the correct conclusion concerning the relationship between the treatment and the results of an experiment.

7.4.1 ***Low statistical power.*** In order to reduce the impact of this threat and mitigate it, we ensured that we had enough data to work with and make correct conclusions. We had a total of 9 PWAs, each undergoing 2 treatments. A total of 20 runs were performed for each trial with a total of 360 runs. Naturally, to achieve even more accurate results in future replications of the experiment, the number of subjects can be increased by redoing the subjects selection phase of our study, while being able to fully reuse our measurement infrastructure.

7.4.2 ***Violated assumptions of statistical tests.*** To mitigate this threat and ensure that the appropriate statistical tests were being used, we firstly checked for the normality of our data by using a combination of various techniques. Then, if the data was normally distributed, and the rest of our assumptions proven to be correct, a t-test would have been applied, otherwise we could migrate to the Wilcoxon signed-rank test.

7.4.3 ***Fishing and error rate.*** This threat does not apply to our empirical experiment. We do not repeatedly perform the tests for significant relationships. The inclusion of outliers and subsequent visualizations prove the absence of fishing for certain results.

8 RELATED WORK

Progressive Web Apps have been investigated only recently, with studies predominantly assessing how they can be used as a more technically sustainable alternative with respect to standard web apps and native mobile apps, specially in terms of multi-platform support, improved user experience, easier operation, better maintainance, etc. [5, 18, 19]. Our research is completely different with respect to the existing literature since we aim at providing *empirical evidence* about selected technical characteristics of PWAs (e.g., caching, energy consumption, performance), so to help developers in taking better informed technical decisions.

To the best of our knowledge, the paper by Malavolta et al. is the only empirical study investigating PWAs and service workers [20]. There, the authors were focussing exclusively on the presence of service workers; specifically, they performed an empirical study targeting the energy efficiency of 7 real PWAs, while having 3 empirical factors: (i) the presence/absence of a service worker in the PWAs, (ii) the network conditions (i.e., 2G vs WiFi), and (iii) the type of mobile device on which the PWAs were running (i.e., low-end vs high-end smartphones). Similarly to our experiment, their experiment did not find enough statistical evidence about the impact that service workers may have on energy consumption (regardless of the network conditions). By assessing the impact of the caching status *within* service workers, our study can be considered as a step further with respect to the research direction initiated by Malavolta et al.

More in general, Pinto and Castor have investigated the main concerns related to the energy efficiency of software applications, such as lack of knowledge or resources to prevent developers from accurately identifying, refactoring, and fixing the high energy consumption hotspots in software applications [22]. From an engineering perspective, small inefficiencies can add up to affect certain components such as battery life, responsiveness, performance of the system and overall application success. Most of the research done so far on correlations between energy efficiency and intensive computational tasks has been heavily concentrated on the lower levels of hardware and software stacks of the application [22]. We recently investigated on the correlation between energy consumption and the performance metrics in the context of mobile web apps [7]. That study is different from ours because (i) it has been designed specifically for investigating the correlation between energy and performance, whereas this study is treating them as separate factors, (ii) it considers a different metrics for the performance dependent variable (i.e., the aggregated performance score produced by Google Lighthouse vs page load time), and (iii) its subjects are generic mobile web apps, whereas this study focusses exclusively on PWAs.

Furthermore, a study conducted on performance based practices that impact the energy efficiency of mobile applications concluded that distinct features of a mobile device, such as battery life, improve significantly with the inclusion of sustainable applications. This emphasizes how important energy efficiency is in software applications [9].

Research on how caching improves web app performance had been done in other works, e.g., [25], where the authors provided evidence about the fact that the use of caching produced several benefits such as reduced latency. Although effective web caching is not a silver bullet, setting up suitable caching policies can provide measurable gains to PWAs with minimal work in the future [16]. However, we were not able to identify studies that focused on the impact the status of the cache may have on the performance of PWAs (in terms of page load time). We hope that our study can inform researchers and web developers on the impact of caching on energy consumption and performance of PWAs, and we hope our results will help developers in better understanding how to improve the caching strategies of their future PWAs.

9 CONCLUSIONS

In this paper we presented the design, conduction, and results of an empirical study assessing the impact of the caching status on the energy consumption and page load time in the context of real-world PWAs. Our results show that there is no evidence that PWAs with empty or populated caches significantly consume different amounts of energy. The results of the experiments also show that overall PWAs load faster with a populated cache. This study provides to researchers and developers empirical evidence about the (medium) gain that PWAs with cached resources can have in terms of page load time.

A possible extension of this experiment is to investigate this phenomenon *in the wild* by inspecting and analyzing the source code of several thousands of real-world PWAs. The goal here is to better understand (i) how developers implement the key enabling features of PWAs such as push notifications, network caching, and background processing and (ii) how they relate with the overall quality of their code, e.g., in terms of performance, energy consumption, maintainability, security. Moreover, we are planning to carry out other follow-up studies focussing on the impact of (i) different caching strategies (e.g., network-first, cache-first, etc.), (ii) different and more realistic network conditions - instead of Wifi only, and (iii) the type and amount of I/O operations performed behind the lines by the browser.

REFERENCES

[1] [n. d.]. How Web Caching Improves Internet Performance. https://www.3pillarglobal.com/insights/how-web-caching-improves-internet-performance. Accessed: September 10, 2019.

[2] [n. d.]. WHAT ARE PWAS? ARE THEY FASTER? ARE THEY A SEARCH RANKING FACTOR? https://www.ezoic.com/what-are-pwas-faster-search-ranking/. Accessed: September 18, 2019.

[3] ApacheBooster. [n. d.]. What is network caching and why do we need it? . https://apachebooster.com/blog/what-is-network-caching-and-why-do-we-need-it/. Accessed: September 20, 2019.

[4] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. 1994. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley.

[5] Andreas Biørn-Hansen, Tim A. Majchrzak, and Tor-Morten Grønli. 2017. Progressive Web Apps for the Unified Development of Mobile Applications. In *Web Information Systems and Technologies - 13th International Conference, WEBIST 2017, Porto, Portugal, April 25-27, 2017, Revised Selected Papers*. 64–86. https://doi.org/10.1007/978-3-319-93527-0_4

[6] Lionel C Briand, Christiane M Differding, and H Dieter Rombach. 1996. Practical guidelines for measurement-based process improvement. *Software Process Improvement and Practice* 2, 4 (1996), 253–280.

[7] Kwame Chan Jong Chu, Tanjina Islam Miguel Morales Exposito, Sanjay Sheombar, Christian Valladares, Olivier Philippot, Eoin Martino Grua, and Ivano Malavolta. 2020. Investigating the correlation between performance scores and energy consumption of mobile web apps. In *Proceedings of the International Conference on Evaluation and Assessment on Software Engineering (EASE)*. ACM, to appear.

[8] Norman Cliff. 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin* 114, 3 (1993), 494.

[9] Luis Cruz and Rui Abreu. 2017. Performance-based guidelines for energy efficient mobile applications. In *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 46–57.

[10] David Novicki. [n. d.]. PWA Asset Caching Strategies. https://codeburst.io/pwa-asset-caching-strategies-8a20c31b2181. Accessed: September 27, 2019.

[11] Jeffrey Erman, Alexandre Gerber, Mohammad Hajiaghayi, Dan Pei, Subhabrata Sen, and Oliver Spatscheck. 2011. To cache or not to cache: The 3G case. *IEEE Internet Computing* 15, 2 (2011), 27–34. Accessed: September 20, 2019.

[12] Mirco Franzago, Henry Muccini, and Ivano Malavolta. 2014. Towards a collaborative framework for the design and development of data-intensive mobile applications. In *Proceedings of the 1st International Conference on Mobile Software Engineering and Systems*. 58–61.

[13] Mihail Gaberov. 2019. How to optimize your JavaScript app by using Service Workers. https://www.freecodecamp.org/news/optimize-your-javascript-app-by-using-service-workers/. Accessed: September 10, 2019.

[14] Robert J Grissom and John J Kim. 2005. *Effect sizes for research: A broad practical approach*. Lawrence Erlbaum Associates Publishers.

[15] Shuai Hao, Ding Li, William GJ Halfond, and Ramesh Govindan. 2013. Estimating mobile application energy consumption using program analysis. In *2013 35th international conference on software engineering (ICSE)*. IEEE, 92–101.

[16] Justin Ellingwood. [n. d.]. Web Caching Basics: Terminology, HTTP Headers, and Caching Strategies. https://www.digitalocean.com/community/tutorials/web-caching-basics-terminology-http-headers-and-caching-strategies. Accessed: September 27, 2019.

[17] Ding Li, Shuai Hao, William GJ Halfond, and Ramesh Govindan. 2013. Calculating source line level energy information for android applications. In *Proceedings of the 2013 International Symposium on Software Testing and Analysis*. ACM, 78–89.

[18] Tim A. Majchrzak, Andreas Biørn-Hansen, and Tor-Morten Grønli. 2018. Progressive Web Apps: the Definite Approach to Cross-Platform Development?. In *51st Hawaii International Conference on System Sciences, HICSS 2018, Hilton Waikoloa Village, Hawaii, USA, January 3-6, 2018*. 1–10.

[19] Ivano Malavolta. 2016. Beyond native apps: web technologies to the rescue!(keynote). In *Proceedings of the 1st International Workshop on Mobile Development*. ACM, 1–2.

[20] Ivano Malavolta, Giuseppe Procaccianti, Paul Noorland, and Petar Vukmirovic. 2017. Assessing the impact of service workers on the energy efficiency of progressive web apps. In *2017 IEEE/ACM 4th International Conference on Mobile Software*

Engineering and Systems (MOBILESoft). IEEE, 35–45.

[21] Maria Gregori, Jesús Gómez-Vilardebó, Javier Matamoros, Deniz Gündüz. [n. d.]. Wireless Content Caching for Small Cell and D2D Networks. https://arxiv.org/abs/1603.04341. Accessed: October 8, 2019.

[22] Gustavo Pinto and Fernando Castor. 2017. Energy efficiency: A new concern for application software developers. *Commun. ACM* 60 (11 2017), 68–75.

[23] Feng Qian, Kee Shen Quah, Junxian Huang, Jeffrey Erman, Alexandre Gerber, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. 2012. Web caching on smartphones: ideal vs. reality. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*. ACM, 127–140. Accessed: September 20, 2019.

[24] S2-Group. 2020. Android Runner. GitHub Repository. https://github.com/S2-group/android-runner

[25] Swaminathan Sivasubramanian, Guillaume Pierre, Maarten van Steen, and Gustavo Alonso. 2007. Analysis of Caching and Replication Strategies for Web Applications. *Internet Computing, IEEE* 11 (02 2007), 60–66. https://doi.org/10.1109/MIC.2007.3

[26] Sayali Sunil Tandel and Abhishek Jamadar. 2018. Impact of Progressive Web Apps on Web App Development. *International Journal of Innovative Research in Science, Engineering and Technology* 07, 09 (2018), 9439–9444.

[27] Sira Vegas. 2018. Analyzing software engineering experiments: Everything you always wanted to know but were afraid to ask. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. 534–535.

[28] C Wohlin, P Runeson, M Höst, M.C Ohlsson, B Regnell, and A Wesslen. 2012. Experimentation in Software Engineering. (2012), 89–101. Accessed: September 22, 2019.

[29] C Wohlin, P Runeson, M Höst, M.C Ohlsson, B Regnell, and A Wesslen. 2012. Experimentation in Software Engineering. (2012), 102–116. Accessed: October 22, 2019.